



# **Meaningful Datasets to Benchmark Facial Recognition Algorithms**

**SCE 17-0601**

**Submitted by: Quah You Seng**

**Matriculation Number: U1621992C**

**Supervisor: Prof Althea Liang Qianhui**

**School of Computer Science & Engineering**

A final year project report presented to the Nanyang Technological University

in partial fulfilment of the requirements of the degree of

Bachelor of Computer Science

**2018**

# Abstract

In recent years, more applications are adopting facial recognition as one of their components, due to the rapid improvement of performance in recognition accuracy and response time. Many algorithms give every good accuracy that their algorithms are implemented as a biometric security system.

Benchmarks like MegaFace and Labeled Faces in the Wild (LFW) test performance of face recognition algorithms with datasets that consist of thousands or millions of images with many complex factors. The images in these datasets are used to test the limits of the algorithms and their performance in dealing with noisy images. The results from these benchmarks are used to justify the performance of that algorithm.

In this project, existing FaceScrub dataset is divided into different datasets according to different factors. Factors include face lighting, posture, skin tone etc. Comparing to other benchmarks, this project aims to benchmark algorithms performance progressively by introducing new factors for each test. By dividing the datasets, results from the benchmark will be more meaningful and improvements can be done to the algorithms accordingly.

Local Binary Pattern Histogram(LBPH) and K-Nearest Neighbour with face\_recognition (K-NN) recognition algorithm were being compared and analysed using the generated datasets. Experiments were done on the recognition algorithms with different datasets and the factors affecting the algorithm were identified.

# Acknowledgements

I will like to express my sincere gratitude to my Final Year Project supervisor, Prof Althea Liang Qianghui for her guidance, valuable suggestions and directions. Prof Liang have been patiently guiding me throughout the Final Year Project, correcting my mistakes and helping me to make improvements on the project. Prof Liang guided through my learning for new knowledge, motivating me to explore more about facial recognition, understanding more about training and testing of data for facial recognition. I truly appreciate Prof Liang for the time and effort given by her.

Quah You Seng

## Table of Contents

1.	Introduction.....	7
2.	Background.....	8
2.1	Project Motivation.....	8
2.2	Project Goal and Objective .....	9
3	Literature Review .....	10
3.1	MegaFace.....	10
3.1	Labeled Faces in the Wild (LFW).....	11
4	Methodology.....	12
4.1	Datasets Preparations .....	12
4.1.1	FaceScrub Dataset.....	12
4.1.2	Flow Diagram of processing images. ....	13
4.1.3	Verifying Images .....	13
4.1.4	Further Processing of images.....	14
4.2	Creating the Datasets.....	15
4.2.1	Dataset 1 & 2 .....	15
4.2.2	Dataset 3 .....	15
4.2.3	Dataset 4 .....	17
4.3	Detection Algorithms .....	18
4.3.1	Haar Cascade Algorithm.....	18
4.3.2	OpenCV's Deep learning detection algorithm.....	21
4.4	Recognition Algorithms .....	23
4.4.1	Local Binary Pattern Histogram (LBPH) Algorithm.....	23
4.4.2	K-Nearest Neighbour Algorithm with <i>face_recognition</i> API.....	24
5	Implementation .....	26
5.1	Benchmark using Python – benchmarker.py .....	26

5.1.1	Method 1 – Algorithm calls benchmarker.py .....	26
5.1.2	Method 2 – benchmarker.py calls the algorithm .....	28
5.1.3	Processing training and testing sets .....	29
5.2	Implementing recognition algorithms to use benchmarker.....	31
5.2.1	Using benchmarker with <i>lbph_algo.py</i> and <i>knn_algo.py</i> .....	31
5.2.2	Training LBPH Algorithm – <i>lbph_algo.py</i> .....	32
5.2.3	Recognizing test images with LBPH algorithm – <i>lbph_predict.py</i> .....	34
5.2.4	Training K-NN Algorithm – <i>knn_algo.py</i> .....	34
5.2.5	Recognizing test images with K-NN Algorithm – <i>knn_predict.py</i> .....	35
6	Experiments and Results.....	36
6.1	Experiment 1 – Face Detection .....	36
6.2	Experiment 2 – Generic Dataset .....	37
6.3	Experiment 3 – Dataset with Clean Faces.....	38
6.4	Experiment 4 – Dataset with Darker Skin Tone .....	39
7	Experiment Findings.....	40
7.1	Face Detection.....	40
7.2	LBPH Algorithm.....	41
7.3	K-NN Algorithm .....	41
8	Conclusions.....	42
9	Recommendations.....	42
	References.....	43
	Program Listing .....	44

Figure 1. Flow Diagram for Processing images .....	13
--	----

Figure 2. An example of detection results for Adam Brody after processing using Haar Cascade Algorithm .....	14
Figure 3. Processing and analysing for Dataset 3 .....	16
Figure 4. Overview of Dataset 5 .....	<b>Error! Bookmark not defined.</b>
Figure 5. Haar Features.....	18
Figure 6. Haar features applied on faces.....	18
Figure 7. Example of face detection using Haar Cascade algorithm.....	20
Figure 8. An example of face detected using deep learning algorithm .....	22
Figure 9. Example of labelling pixels of a 3 x 3 pixels .....	23
Figure 10. LBPH extracting histogram.....	24
Figure 11. Illustration of K-NN Algorithm .....	25
Figure 12. Sequence diagram of Algorithm calling Benchmark .....	27
Figure 13. Dynamic importing of algorithm and menu display .....	28
Figure 14. Checking of answer for Method 1 .....	30
Figure 15. Source code to import benchmarker and fetching images (Method 1). ...	31
Figure 16. Example of both functions by the algorithm for benchmarker .....	32
Figure 17. Code for image normalization.....	33
Figure 18. Code for resizing of image .....	33
Figure 19. Prediction of LBPH algorithm. ....	34
Figure 20. Code for K-NN predictor .....	35
Figure 21. Results of Experiment 1 .....	36
Figure 22. Results for Experiment 2 .....	37
Figure 23. Results for Experiment 3 .....	38
Figure 24. Results for Experiment 4 .....	39
Figure 25. Haar Cascade Detection vs Deep Learning Detection .....	40

## 1. Introduction

Facial recognition is becoming one of many key features for computers or handheld devices these days. Smartphone and tablet companies, such as Apple and Samsung, introduced their products using facial recognition as one of their main key features for securing devices. For example, Apple claims that the new iPhone X FaceID is “a revolution in recognition” and faces can be used as password to unlock, authenticate and make payments. These companies are constantly trying to improve their facial recognition algorithms to stay competitive in the market.

The demand of integrating facial recognition technology into other software application is also on the rise. Facebook uses facial recognition to record the face of users and give suggestions on who to tag and automatically tagging users after uploading pictures. The Singapore government uses facial recognition for security purposes such as Singapore’s “Smart Base Access” solution, which expedite the screening process by recognising visitors’ faces when they enter Singapore army bases [1]. Since facial recognition is used for both commercial and security purposes, algorithms must have fast response time and provide high accuracy in recognition. Response time can be measured by including timers during recognition but what about the accuracy of the algorithms?

## 2. Background

### 2.1 Project Motivation

There are many different types of facial recognition algorithms in the world. From the more commonly known Local Binary Pattern Histogram (LBPH) algorithm, which gives decent accuracy, to deep learning algorithms. Studies have been conducted [2] comparing different algorithms and their performance. A study conducted by Alice et al [3] stated that the United States Government funded competitions to compare facial recognition algorithms to encourage programmers to create their own algorithms or to improve the performance of existing algorithms. To ensure fairness, the competition uses the same dataset for all participating algorithms and compare the performance for each algorithm.

Comparing algorithms by hosting competitions is not as efficient because competitions are not hosted often. Algorithm programmers are constantly trying to improve their algorithms but will have to wait for competitions to be hosted to see where they stand. To address this issue, benchmarks such as MegaFace and Labeled Faces in the Wild (LFW) are designed to allow algorithm programmers to test their algorithms and compare performance with existing known algorithms at their own work stations.

MegaFace and LFW provides their own datasets which have images of many identities with unconstrained pose, expressions, lightings, and exposure. Both datasets provide very good images for algorithms to test on. Datasets provided consist of a mixture of different factors, stressing the algorithms to their limit. However, some algorithms can perform better in certain conditions as compared to algorithms that work well in general. Using existing benchmarks to assist in developing an algorithm may cause the programmers to “guess the factors” that may be causing the difference of even 1% accuracy of their algorithms. For instance, Algorithm A may work well in recognising images with normal lighting but not so well in recognising images with a mixture of different lighting.



## 2.2 Project Goal and Objective

The goal of this project is to use and analyse existing datasets of images, classifying them into different groups of images according to the different factors (lighting, face angle etc.), and to create a more meaningful dataset to benchmark facial recognition algorithms.

Our objective is to reduce the “guessing” process during development of algorithms and to test how algorithms work with images of different factors. Programmers will be able to test their algorithms with different datasets and get a better understanding on how their algorithm works. These datasets can also be an important guide to help new programmers who are picking up facial recognition.

### 3 Literature Review

Benchmarks such as MegaFace and Labeled Faces in the Wild (LFW) create their own datasets and use it to set up challenges for developers to train and test their algorithms.

#### 3.1 MegaFace

MegaFace [4] [5] has a database of 1 Million photos of more than 690K identities, which are carved from Flickr. It evaluates performance by inserting “distractors” into the gallery set (Probe set). Algorithms that performed above 95% accuracy on LFW can only achieve 35%-37% identification rates with 1 Million distractors. Probe sets used by MegaFace are FGNet and FaceScrub. MegaFace uses their own datasets as distractors, which objective is to stress and confuse recognition algorithms. Dataset includes images of people with different expressions, pose, lighting, and exposure, to allow a good mixture of datasets. FaceScrub includes 100K photos of 530 celebrities, where MegaFace will randomly select 40 males and 40 females who have more than 50 images per person. FGNet is an aging dataset which includes 975 images of 82 identities, where each identity includes images of different ages of the person.

MegaFace tests consist of verification and identification. Verification provides a pair of photos for the algorithm and the algorithm should verify if both images consist of the same person. Identification provides a set of photos(probe) and a gallery of photos. The algorithm needs to use the probe set to find the images of that person in the gallery.

MegaFace consists of 2 challenges (MegaFace Challenge 1 and Challenge 2). Challenge 1 inserts “distractors” into the gallery set and perform evaluation by increasing the number of “distractors” (10, 100, 10000, ..., 1M) at each test. The test is to determine the accuracy of the algorithm with the increasing number of distractors.

Challenge 2 is for deep learning facial recognition algorithms as these algorithms are out-performing traditional algorithms. Challenge 2 acts similarly to Challenge 1, except that it provides 4.7M photos of 672K identities for algorithms to train before starting the challenge.

### 3.1 Labeled Faces in the Wild (LFW)

LFW [6] provides 13,223 images with 5,749 identities, with images being labelled. Text files are provided, which store the pairs matched (both training and testing), number of images per identity. Originally, LFW has 2 configurations, which are restricted configuration and unrestricted configurations. The difference between restricted and unrestricted configuration is that restricted configuration requires the algorithm to strictly follow the pair.txt while unrestricted configuration does not. For example, Image A = B and C = D are used for training for Leo, in unrestricted configuration, the algorithm can put A = C and A = D into the training to increase the amount of training data for the algorithm.

Due to the need of data in an unsupervised setting and using outside data for training, LFW now provides six protocols for training data. This includes:

- Unsupervised
- Image-restricted with no outside data
- Unrestricted with no outside data
- Image-restricted with label-free outside data
- Unrestricted with label-free outside data
- Unrestricted with labelled outside data

Similarly, LFW test algorithms using identification and verification.

Both MegaFace and LFW focus on testing the limits of algorithms by using images with complex factors. In this project, we will focus on developing a more meaningful dataset by dividing existing dataset into different factors and testing the performance of algorithms.

## 4 Methodology

### 4.1 Datasets Preparations

Datasets preparation consist of verifying and processing the original FaceScrub dataset into 5 different datasets, which are as followed:

1. Dataset 1 – consist of 50 identities from Haar Cascade algorithm processed dataset.
2. Dataset 2 – consist of 50 identities from OpenCV's Deep Learning Detector processed dataset.
3. Dataset 3 – consist of 20 identities (10 males and 10 females) with clean faces.
4. Dataset 4 – consist of 21 identities with darker skin tone.

Datasets 1 to 4 consist of 1 training set and 1 test set to benchmark the performance of algorithms.

#### 4.1.1 FaceScrub Dataset

For this project, images from FaceScrub [7] will be extracted to create new datasets. FaceScrub consists of 106,863 face images of 530 celebrities. Face images were extracted from the internet and images were taken under uncontrolled conditions. FaceScrub is used for this project as it provides many images per identity to train and test facial recognition algorithms.

#### 4.1.2 Flow Diagram of processing images.

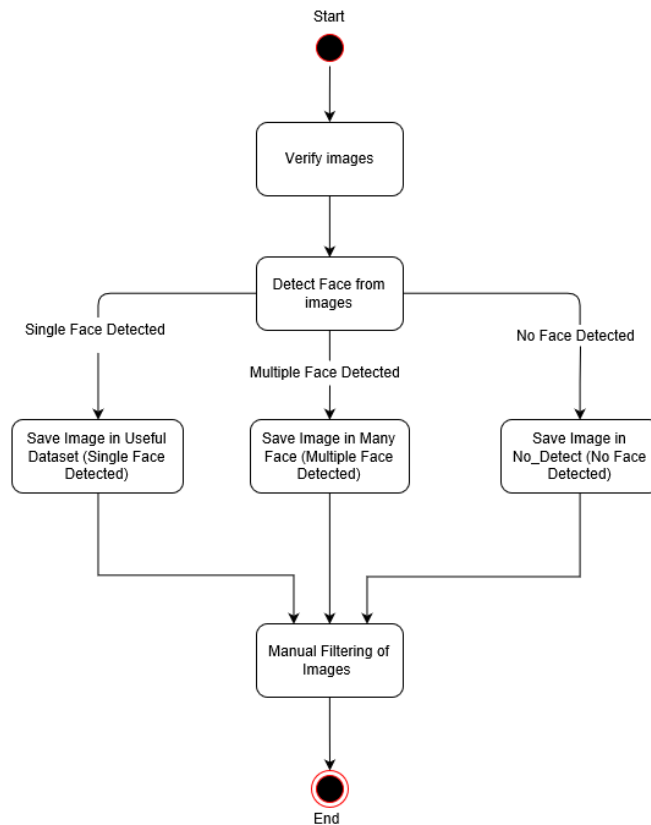


Figure 1. Flow Diagram for Processing images

#### 4.1.3 Verifying Images

Some images extracted from FaceScrub are corrupted and could not be opened. Therefore, we need to verify all images and remove corrupted images.

```
try:
    img = Image.open(path)
    img.verify()
    #print("Valid image")
except Exception:
    os.remove(path)
    print(path + "is removed")
    continue
```

The code above tries to open the image file and verify if it is an actual image. If the image cannot be opened and verified, an exception will be returned, and the corrupted file will be removed. After verifying all images, 95,961 of the 106,863 images are valid and will be used for further processing.

#### 4.1.4 Further Processing of images

Face detection is a crucial step before facial recognition. In the later part of the report, an experiment will show that using a better face detection algorithm can affect the results of face recognition. For this project, we will be using 2 different face detection algorithms, the Haar Cascade algorithm and OpenCV's deep learning face detection. With reference to figure 1, we use the verified dataset to generate 2 different datasets by using face detection algorithms to categorize them into different folders.

Images will be sorted into "Useful Dataset", "No\_Detect", and "Many Face", according to the results of each detection algorithm (refer to figure 2). "Useful Dataset" consist of images where the algorithms detect a single face, "No\_Detect" consist of images where algorithms detect no faces, and "Many Face" consist of images where many faces are detected within the image.

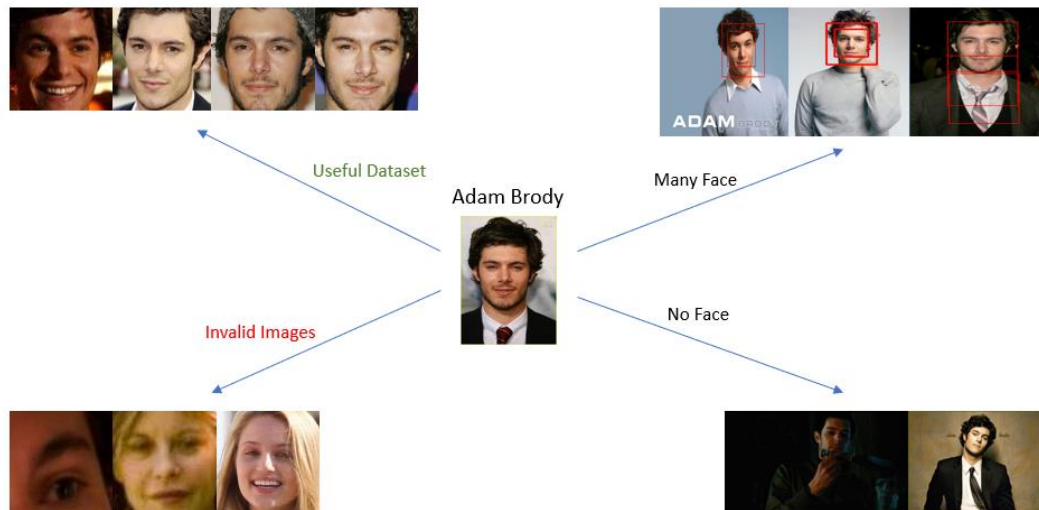


Figure 2. An example of detection results for Adam Brody after processing using Haar Cascade Algorithm

Faces detected need to be manually filtered as there may be cases where the face of the identity is not detected while another person in the image is detected. Random part of the image can also be mistakenly detected as a face (refer to figure 2).

## 4.2 Creating the Datasets

### 4.2.1 Dataset 1 & 2

Dataset 1 consist of 6,020 images of the first 50 identities from the “Useful Dataset” generated using the Haar Cascade detection algorithm. Dataset 2 consist of 6,925 images of the first 50 identities from “Useful Dataset” generated using OpenCV’s deep learning detection. Each image has been manually scanned through again to ensure that there is no incorrect faces detected.

The main goal for Datasets 1 & 2 is to create a baseline of images for face detection algorithms that developers can use. Both datasets consist of images with complex factors. Dataset 1 is suitable for programmers who use the Haar Cascade algorithm, which most facial recognition beginners use, to detect faces. Dataset 2 can be the baseline for recognition algorithms using deep learning detection. Although both are baselines for detection algorithms, other detection algorithms can still be used on these images.

In this project, Datasets 1 and 2 will be used to test the performance of 2 different recognition algorithms on complex images and conduct an experiment to find out if face detection can affect recognition results.

### 4.2.2 Dataset 3

Dataset 3 includes 1,310 images of 10 males and 10 females with at least 50 images per identity. The images were carefully selected by finding frontal faces (faces that are facing the front). The face region of the image should not be covered by hands, hair fringe or any objects, and the lighting should be consistent throughout the entire face. The dataset consists of identities with different skin tone and different ages.

The motivation for Dataset 3 is to test the algorithms how they perform in a “purer” dataset by using only frontal faces and not faces facing different directions. Having consistent lighting throughout reduces the possibility of lighting being a factor that may affect performance of recognition algorithms.

The difficulty of generating Dataset 3 is determining the consistency of lighting throughout the entire face. This is because it is hard to spot the small differences in lighting within certain images, especially for images with darker lighting. Therefore, to better determine this factor, we crop the faces from images and convert it into grayscale.

Grayscale images will be analysed for consistent lighting on the face. The direction of where the face is facing will be checked. Images with frontal face and consistent lighting throughout the face will be valid for this dataset. Images with inconsistent lighting or non-frontal faces will be rejected for Dataset 3(Refer to figure 3)

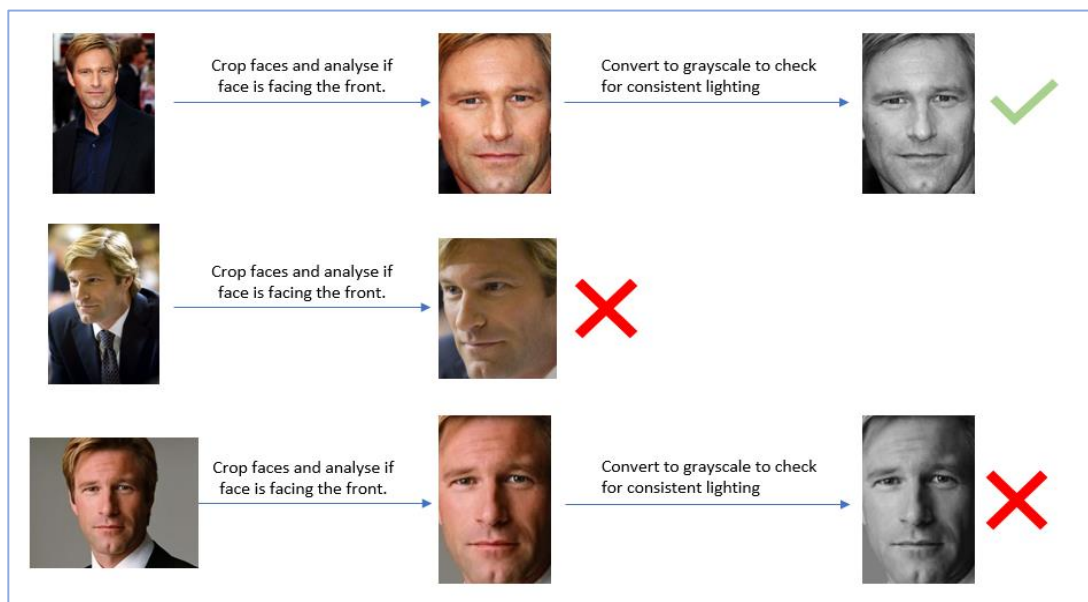


Figure 3. Processing and analysing for Dataset 3



#### 4.2.3 Dataset 4

Dataset 4 consist of 21 identities with a total of 2,465 images. Dataset 4 includes identities who have darker skin tone. This is a concern as there are facial recognition algorithms that have trouble detecting or recognizing faces of people with darker skin tone. For example, facial recognition systems designed by Microsoft, IBM, and Megvii produced up to 12% gender misidentification rate for darker-skinned males and up to 35% for darker-skinned females [4]. This means that skin tone can be a factor that may affect recognition algorithms.

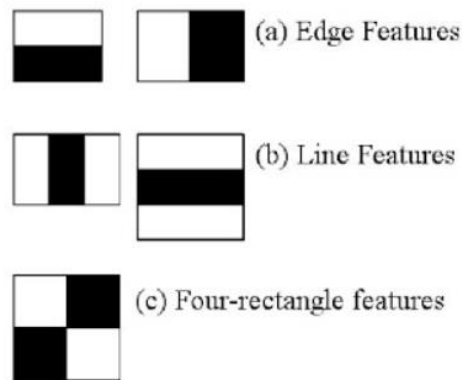
To further process the dataset, images with faces which are partially covered (e.g wearing sunglasses, hand covering over face.) will be rejected. Images collected are frontal faces.

### 4.3 Detection Algorithms

Performance of Local Binary Pattern Histogram (LBPH) and K-Nearest Neighbour(K-NN) with face\_recognition API will be tested on the datasets. Both algorithms follow the same steps when evaluating performance. The steps are face detection, label the faces, train the algorithms, and recognition testing.

#### 4.3.1 Haar Cascade Algorithm

Haar Cascade algorithm is a machine learning based approach for object detection proposed by Paul Viola and Michael Jones. Both negative and positive images are required to train the cascade functions. After training, features are extracted by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle, using the haar features shown in figure 5.



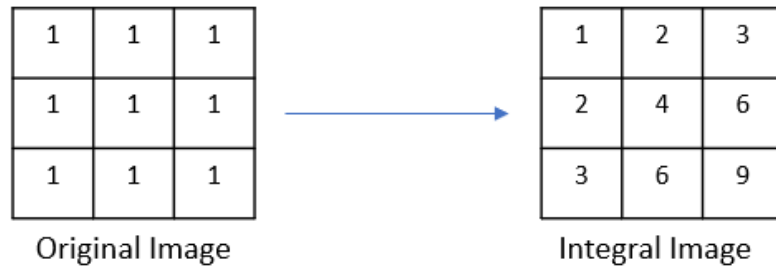
*Figure 4. Haar Features*

We can apply the haar features onto an image to find different features. As human share some similar properties, these haar features will be used to seek for these properties to determine if the region is a pair of eyes, nose, etc (refer to figure 6). An example of a common property of humans is our nose bridge being brighter than our eyes.

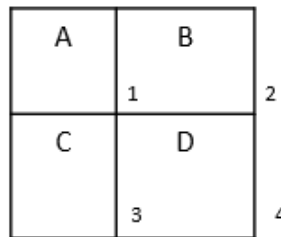


*Figure 5. Haar features applied on faces*

To speed up the calculation process of summing up pixels, we can use integral image. In an integral image, the value of the pixel is the sum of pixel above and to the left of that pixel.



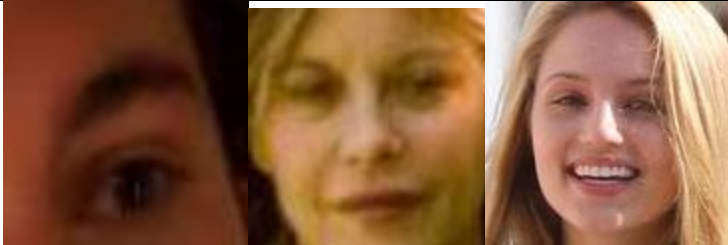

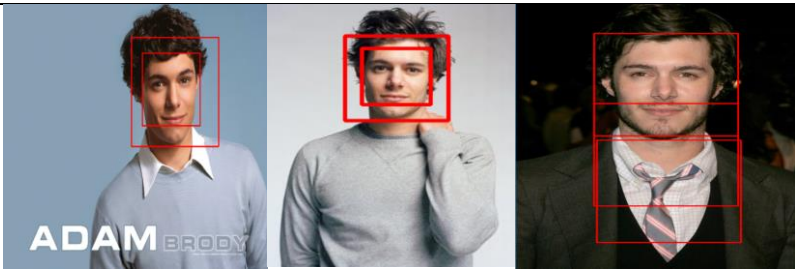


By converting to integral image, we can speed the sum of pixel by doing 4 calculations.



$$\begin{aligned}
 \text{Sum of pixel in D} &= 1 + 4 - (2 + 3) \\
 &= A + (A+B+C+D) - (A+C+A+B)
 \end{aligned}$$

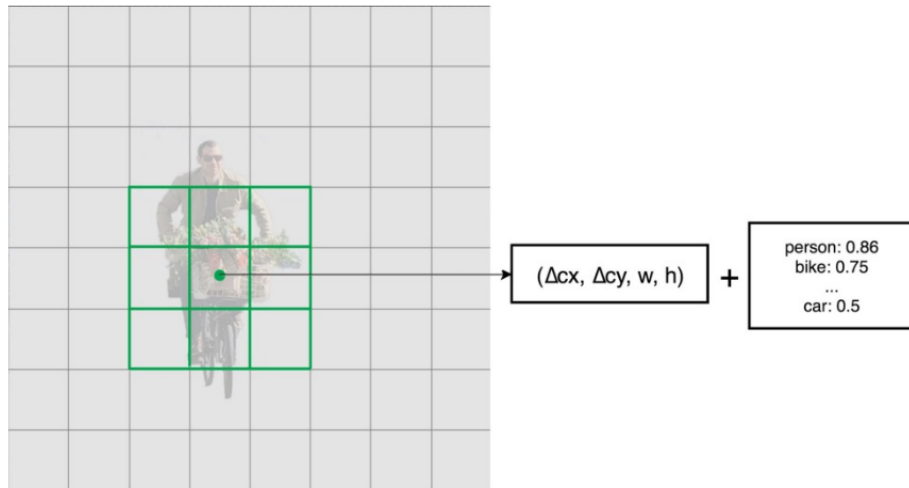
To further reduce the processing time, Adaboost is used to remove irrelevant features by giving weight to each feature. Linear combination of these features will determine if the region is a face. Cascading is used so that each cascade window will classify different features and if all windows succeed in finding features, a face is detected. Figure 7 shows an example of Haar Cascade used on the datasets used in the project.

<p align="center"><b><u>Haar Cascade Algorithm</u></b></p> <p align="center">Adam Brody</p>  <p align="center">Number of total images: 194</p> <p align="center">Number of single faces: 124</p> <p align="center">Number of no face: 18</p> <p align="center">Number of multiple faces: 35</p>	
Single face detected (Valid):	
Single face detected (invalid):	
No face detected:	
Multiple face detected:	

*Figure 6. Example of face detection using Haar Cascade algorithm*

#### 4.3.2 OpenCV's Deep learning detection algorithm

The deep learning algorithm is based on Single Shot Detector(SSD) Framework with ResNet base network. SSD is designed for real-time object detection. SSD object detection consist of extract feature maps and apply convolution filters to detect objects [5]. Each prediction consists of a boundary box and a score of 21 for each class (with an additional class for no object detection).



SSD uses VGG16 architecture to extract feature maps and uses Conv4\_3 layer to detect objects. Small convolution filters are used to compute location and class score of objects detected. After extracting feature maps, 3 X 3 convolution filters are applied for each cell to make predictions and 4 predictions are made for each cell. The results of detection will be the class with the highest score.

In this project, the OpenCV's deep learning detection algorithm will be used as it returns a better crop of a face image as compared to Haar cascade algorithm (refer to figure 7 and 8).

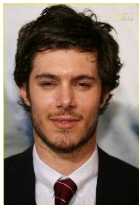

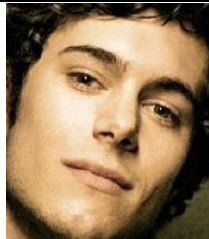

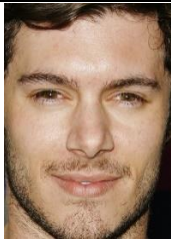





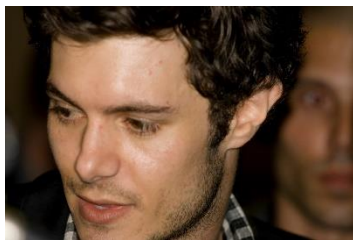


<p style="text-align: center;"><b><u>Deep Learning Algorithm</u></b></p> <p style="text-align: center;">Adam Brody</p>  <p style="text-align: center;">Number of total images: 194</p> <p style="text-align: center;">Number of single faces: 149</p> <p style="text-align: center;">Number of no face: 7</p> <p style="text-align: center;">Number of multiple faces: 21</p>	
Single face detected (Valid):	    
Single face detected (invalid):	-
No face detected:	   
Multiple face detected:	  

Figure 7. An example of face detected using deep learning algorithm

## 4.4 Recognition Algorithms

### 4.4.1 Local Binary Pattern Histogram (LBPH) Algorithm

Local Binary Pattern(LBP) is a texture operator that describes texture and shape of a digital image. LBP labels the pixels of an image by thresholding the neighbourhood of each pixel and outputs a binary number. LBP then uses histograms to represent face images with a simple data vector.

To get the value for each pixel, we need to first convert images into grayscale. After converting, each pixel will have a value according to pixel intensity. The value ranges from 0 (black pixel) to 255 (white pixel).

A neighbourhood of different sizes surrounding a centre pixel will be selected. The centre pixel will act as a threshold to compare with neighbourhood pixels. Neighbourhood pixels will be marked as 1 if the pixel has intensity value larger than centre pixel, and 0 if the value is smaller than centre pixel. The result will be a single decimal value labelled in the centre pixel which is calculated by consolidating the binary value of the neighbourhood pixel (refer to figure 9).

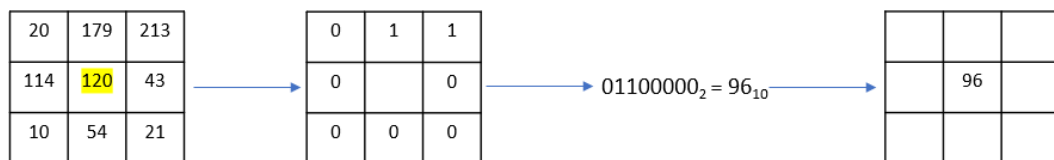


Figure 8. Example of labelling pixels of a 3 x 3 pixels

The decimal number is the value of LBP. Since there are 8 neighbourhood pixels for each centre pixel, there will be a total of  $2^8 = 256$  different patterns.

To extract the histograms, grayscale images will be divided into multiple grids. Each grid will produce a 256-bin histogram and all histograms will be combined into one concatenated histogram (refer to figure 10).

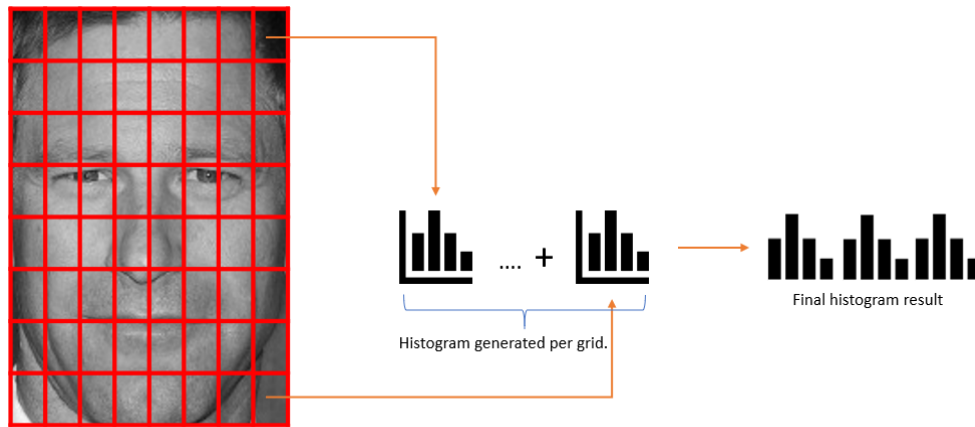


Figure 9. LBPH extracting histogram

#### 4.4.2 K-Nearest Neighbour Algorithm with *face\_recognition* API

K-Nearest Neighbour(K-NN) is a form of instance-based learning where there are no computation or assumption done to the data until classification. K-NN algorithm is also considered a lazy algorithm as it does not use training data points to do generalization. This means that K-NN keeps all training data, which will be used during testing phase.

K-NN for face recognition is used for classification and it groups images into different classes based on their discrete value. The classification of images is made by voting. The image will be classified as the identity (class) most common to its neighbour. The “K” in K-NN is the number of nearest neighbours. An illustration of K-NN algorithm is shown on figure 11.



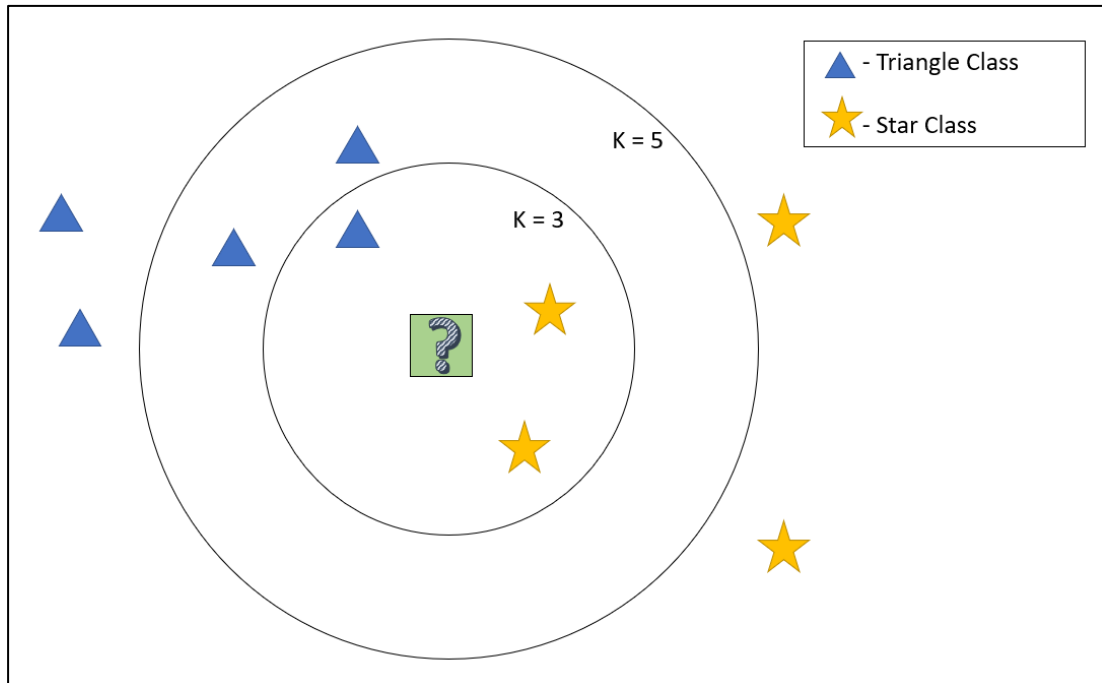


Figure 10. Illustration of K-NN Algorithm

For example, when  $K = 3$ , we will get the discrete value of the new Image A and find 3 nearest neighbours (trained images) and do a vote. If 2 out of 3 existing images is classified as Star, then Image A belong to Star class. However, the number of neighbours can affect results. For instance, Figure 11 shows that the new image will be classified as a Star when  $K = 3$  and classified as a Triangle when  $K = 5$ .

The discrete value for this algorithm will be generated by face\_recognition Python library. We will utilise `face_recognition.face_encoding(image)` function provided by Python Library. This function will return a 128-dimension (128 real numbers) face encoding for each face in the image.

## 5 Implementation

This project is developed using Python 3.5.4 and OpenCV 3.3.0.

### 5.1 Benchmark using Python – *benchmarker.py*

Recognition programmers can use *benchmarker.py* to access the datasets for training and testing. Users can use their algorithm to call methods from the *benchmarker.py* to request for training and testing images. Alternatively, *benchmarker.py* allow users to dynamically import their algorithm files to test on the datasets.

#### 5.1.1 Method 1 – Algorithm calls *benchmarker.py*

Algorithm programmers can import *benchmarker.py* into their codes and call functions to fetch the training and testing images.

The functions descriptions are as follow:

- `fetchTrainingData(DS_DIR)`
  - Use to request training set images with label names of the dataset
  - Parameters required:
    - `DS_DIR` – Name of dataset folder
  - Returns include:
    - `imageArr` – A list of images (each as numpy array)
    - `labelArr` – A list of label names (associated with `imageArr`)
    - `DS_DIR` – Name of dataset folder
- `fetchTestQuestion()`
  - Use to request test questions from *benchmarker.py*
  - Returns include:
    - `test_set` – A list of images (each in numpy array)
- `submitAnswer(ansArr)`
  - Use to submit answer in responds to the test questions
  - Parameter required:
    - `ansArr` – A list of label names (in responds to `test_set`)
  - Return include:
    - `correctAns` – Number of correct recognitions
    - `wrongAns` – Number of wrong recognitions
    - `acc` – Accuracy of algorithm

Programmers are required to call the methods sequentially or an error message will be prompted. For better understanding, figure 12 shows the sequence diagram between the algorithm and benchmarker.

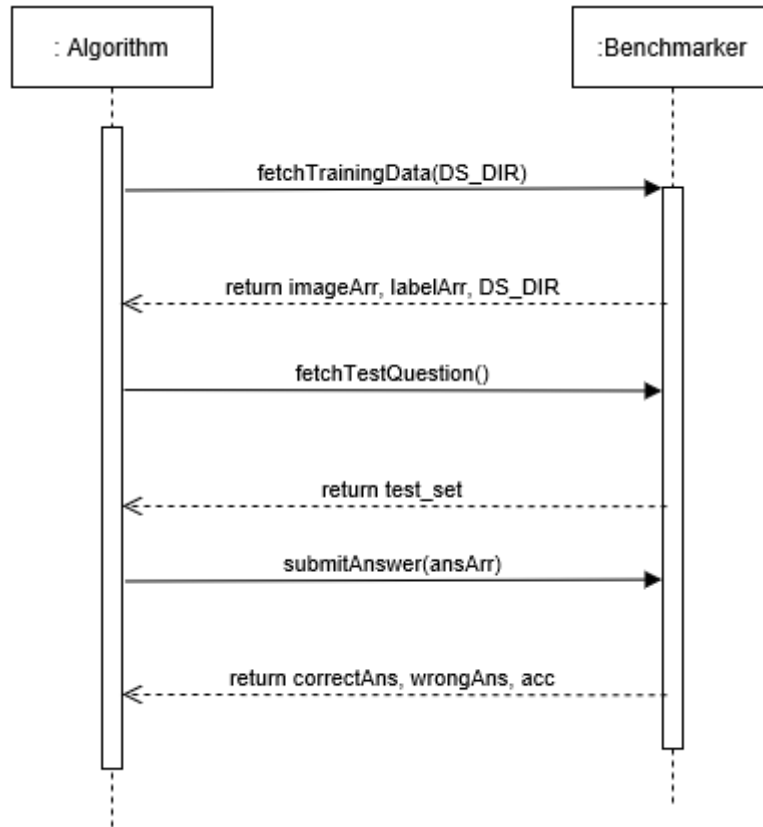


Figure 11. Sequence diagram of Algorithm calling Benchmarker

However, Dataset 5 requires programmers to call `fetchTrainingData(DS_DIR)` another time to do Test 2 and Test 3.

The sequence of calling benchmarker to use dataset 5 is as follow:

1. `fetchTrainingData(DS_DIR)` // fetch Training-Pure
2. `fetchTestQuestion()` // fetch Test 1
3. `submitAnswer(ansArr)` // submit answer for Test 1
4. `fetchTrainingData(DS_DIR)` // fetch Training-Mix
5. `fetchTestQuestion()` // fetch Test 2 - Angle
6. `submitAnswer(ansArr)` // submit answer for Test 2
7. `fetchTestQuestion()` // fetch Test 3 - lighting
8. `submitAnswer(ansArr)` // submit answer for Test 3

Programmers are not restricted to calling all methods for dataset 5 to work but methods must be called sequentially.

### 5.1.2 Method 2 – benchmarker.py calls the algorithm

The benchmarker allows user to import algorithm file dynamically, calling the functions of the algorithm file to pass images for training and testing. This allows the benchmarker to take control of the training and testing process. A few functions are required by the algorithm for the benchmarker to work.

The descriptions of functions required are as follow:

- trainAlgo (imageArr, labelArr, DIR\_NAME)
  - Use to pass training images and label names to the algorithm
  - Parameters required:
    - imageArr – A list of images (each as numpy array)
    - labelArr – A list of label names (associated with imageArr)
    - DIR\_NAME – Name of dataset folder
- testAlgo(image, DIR\_NAME)
  - Use to allow benchmarker to call algorithm to get the identity of the image
  - Parameters required:
    - image – A single image (as numpy array)
    - DIR\_NAME – Name of dataset folder
  - Expected returns:
    - name – label name of identity in the image

To use the benchmarker, programmers will have to key in the algorithm file name after executing benchmarker.py. A menu will be displayed for easy selection of datasets (refer to figure 13). The benchmarker will automatically send training images, test images and generate results.

```
D:\LeoQ\Desktop\FYP Final>python benchmarker.py
Key in your ALGORITHM file name (without .py): lbph_algo
Import ALGORITHM FILE successful
Dataset to generate Train and Test Set:
1) Dataset 1
2) Dataset 2
3) Dataset 3
4) Dataset 4
5) Dataset 5
0) Exit
Your Choice:
```

Figure 12. Dynamic importing of algorithm and menu display

### 5.1.3 Processing training and testing sets

It is the benchmarker's task to sort all training images into an array with their labels saving in another. The training image array is created by accessing each identity folder, looping through all images and saving their name (folder name) into label array. An example is as shown below:

```
imgArr      = [A1, A2, A3, B1, B2, B3, .....]
```

```
labelArr     = ["Adam", "Adam", "Adam", "Jay", "Jay", "Jay", .....]
```

\*Note that each item in imgArr are in numpy array

When testing recognition algorithms, test images will be randomly selected from test set folder. All images of all identities will be tested in a random manner. Numpy array of the image is passed to the algorithm instead. Directories of the image should not be known to the algorithm due to possible case of cheating by reading the directory.

It is also the responsibility of the benchmarker to check the answers from the algorithms. For method 1, the benchmarker will save all labels in a test\_Question array. This array saves the label name corresponding to the test set given to the algorithm. After submitAnswer(ansArr) is called, the benchmarker will check the ansArr with test\_Question to generate the results of the recognition (refer to figure 14).

```

def submitAnswer(ansArr):
    global test_Question
    global test_Dir
    correctAns = 0
    wrongAns = 0

    if not test_Question:
        print("Please fetch the test questions before submitting")
    else:
        BASE_DIR = os.path.dirname(os.path.abspath(__file__))
        wrong_dir = os.path.join(BASE_DIR, "Incorrect Answer")

        if not os.path.exists(wrong_dir):
            os.makedirs(wrong_dir)
        else:
            shutil.rmtree(wrong_dir)
            os.makedirs(wrong_dir)

        print("*** Checking your answer ***")
        for x in range(len(test_Question)):
            print("Question: " + test_Question[x].replace(" ", "-").lower())
            if ansArr[x] is not None:
                print("Answer: " + ansArr[x].replace(" ", "-").lower())
            else:
                print("Answer: ")
            print("")
            if ansArr[x] is not None and ansArr[x].replace(" ", "-").lower() == test_Question[x]:
                correctAns += 1
            else:
                wrongAns += 1
                copyDir = str(x) + " Qn-" + test_Question[x] + " Ans-" + ansArr[x].replace(" ", "-").lower()
                shutil.copyfile(test_Dir[x], (wrong_dir + "\\ " + str(copyDir)))

        print("No of correct: " + str(correctAns))
        print("No of wrong: " + str(wrongAns))
        acc = (correctAns / (correctAns + wrongAns)) * 100
        print("Accuracy is " + str(acc) + "%\n")
        return correctAns, wrongAns, acc

```

Figure 13. Checking of answer for Method 1

For method 2, the benchmarker finds a random test image from test set folder and calls `testAlgo(image)` with single image as parameter, checking the answer returned. The benchmarker will not create array for images to save computational time and memory.

Wrongly recognised images will be saved in a "Incorrect Answer" folder for algorithms to know which images are incorrectly recognised. Source code for benchmarker will be listed in program listing.

## 5.2 Implementing recognition algorithms to use benchmarker

### 5.2.1 Using benchmarker with *lbph\_algo.py* and *knn\_algo.py*

Both algorithm files can interact with the benchmarker using methods 1 and 2. When executed, benchmarker.py will be imported. The codes in figure 15 show that importing is done during runtime. However, programmers can simply just put an import statement at the start of the file to import during execution time.

```
def main():
    pythonFile = "benchmarker"
    nameList = []
    nameList.clear()
    try:
        bm = importlib.import_module(pythonFile, ".")
        menu = True
        spam_info = imp.find_module(pythonFile)
        print("Import Benchmark successful")

        ##### Dataset 1-4#####

        DS_DIR = "Dataset 4"
        imageArr, labelArr, DS_DIR = bm.fetchTrainingData(DS_DIR)
        print("Run Successful")
        trainAlgo(imageArr, labelArr, DS_DIR)

        imgArr = bm.fetchTestQuestion()
        #print(len(imgArr))
        for i in range(len(imgArr)):
            name = testAlgo(imgArr[i], DS_DIR)
            nameList.append(name)

        bm.submitAnswer(nameList)
```

Figure 14. Source code to import benchmarker and fetching images (Method 1).

Both algorithm files import benchmarker and follows the sequence of function call stated in method 1 of benchmarker. The function `fetchTrainingData(DS_DIR)` is called to collect the list of images with their labels. These images and labels will then be trained by the algorithm by calling its own `trainAlgo(imageArr, labelArr, DS_DIR)`. Similarly, `fetchTestQuestion()` returns a list of test images and algorithm will have to test using these images and submit the answer back to the benchmarker.

Alternatively, the benchmarker can dynamically import the algorithm files. `Lbph_algo.py` and `knn_algo.py` both have the methods available for benchmarker to pass training images and to test the algorithm, as shown in figure 16.

```
def testAlgo(image, DS_DIR):
    if at.get_yamlfile() is None:
        at.set_yamlfile(DS_DIR)

    if at.get_pickFile() is None:
        at.set_pickFile(DS_DIR)

    return at.lbph_pred(image)

def trainAlgo(imageArr, labelArr, DIR_NAME):
    proto = "ML/deploy.prototxt.txt"
    caffmodel = "ML/res10_300x300_ssd_iter_140000.caffemodel"
    confid = 0.7
```

Figure 15. Example of both functions by the algorithm for benchmarker

When these methods are called, the algorithm should proceed with their necessary training and testing of the data provided.

### 5.2.2 Training LBPH Algorithm – `lbph_algo.py`

The first step of training LBPH algorithm is to crop the face region from the image using face detection. The crop image (in numpy array) will be converted to grayscale image and added into an image array. To prevent different identity of same name to be trained and classified under the same person, a python dictionary is used to assign a label name to an ID. The ID will be appended into the label array instead of the label name.

Images are normalized to equalize the histogram. This is done to get better distributions of intensities, gaining contrast for lower contrast areas, making face features more distinct for algorithms to learn (refer to figure 17).



```
def normalize_img(images):
    images_normalized = []
    for image in images:
        try:
            if len(image.shape) == 3:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                images_normalized.append(cv2.equalizeHist(image))
        except Exception as e:
            pass
    return images_normalized
```

Figure 16. Code for image normalization

After image normalization, images will be resized to standardize the size of images used to train and test the algorithm (refer to figure 18).

```
def resize(images, size = (100,100)):
    images_norm = []
    for image in images:
        if image.shape < size:
            image_norm = cv2.resize(image, size, interpolation = cv2.INTER_AREA)
        else:
            image_norm = cv2.resize(image, size, interpolation=cv2.INTER_CUBIC)

        images_norm.append(image_norm)
    return images_norm
```

Figure 17. Code for resizing of image

The resized images will then be trained by the LBPH algorithm by passing the image array and label array as input parameters. Label dictionary will be saved into a pickle file and the LBPH recognizer will be saved in a .yaml file. Source code for *lbph\_algo.py* will be shown in the program listing.

### 5.2.3 Recognizing test images with LBPH algorithm – *lbph\_predict.py*

Before recognizing the test images, trained .yml file and pickle file (stores label names) needs to be read. The face of the test image will be cropped using face detection and convert cropped image to grayscale. Normalizing and resizing are done before algorithm does its prediction.

During recognition, the LBPH predictor will return a label and a confidence value of the image recognized. The label we used for training are label ID and will be compared with the name dictionary stored in the pickle file to get the name of that identity. The confidence refers to how certain the algorithm thinks of the identity to the test image. The lower the confidence value, the higher the confidence level. In this project, we set confidence to be lower than 120 (refer to figure 19). Any value higher than 120 will be set as unknown identity. Source code for *lbph\_predict.py* will be shown in the program listing.

```
id_, conf = self.recognizer.predict(roi_gray[0])
print("confidence: " + str(conf))
if conf >= 0 and conf <= 120:
    name = self.labels[id_]
```

Figure 18. Prediction of LBPH algorithm.

### 5.2.4 Training K-NN Algorithm – *knn\_algo.py*

For K-NN algorithm, we will be using Python's *sklearn.neighbours* to generate the ball tree data structure to partition images in different hyperspheres. Unlike LBPH algorithm, there is less processing needed to be done before training the image. During training, faces will be cropped using face detection and cropped images will be passed to face\_recognition library to get 128-dimension number value of face encoding. These 128-d values will be appended into an array along with the label names of each training image.

Both arrays will be processed by sklearn's KNeighbourClassifier to classify each image according to their label names. We will represent each image trained as a node in the ball tree. Distance is used as weights to compare between nodes. The value of weight is inverse to their distance. This means that the higher the distance, the lower the weight. The closest node will have the largest weight. The trained classifier will be stored in a pickle file. Source code for knn\_algo.py will be shown in the program listing.

#### 5.2.5 Recognizing test images with K-NN Algorithm – knn\_predict.py

Like LBPH algorithm, trained pickle will need to be read before recognizing test images. Face detection is made on the test image and the face region is cropped. The cropped image will be passed to face\_recognition to get the 128-d value of the test image. A threshold distance is set to 0.6. The 128-d value of test image will be compared with the existing trained data and will be classified as the identity of the nearest 1 neighbouring node within the threshold distance. If no nodes are found within 0.6 distance radius, the identity of the test image will be classified as "Unknown" (refer to figure 20). Source code for knn\_predict.py is shown in program listing.

```
try:
    faces_encodings = face_recognition.face_encodings(image[startY:endY, startX: endX])
    #print(len(faces_encodings))
    # Use the KNN model to find the best matches for the test face
    closest_distances = self.knn_clf.kneighbors(faces_encodings, n_neighbors=1)
    are_matches = [closest_distances[0][0][0] <= self.distance_threshold]
    # Predict classes and remove classifications that aren't within the threshold
    for pred, rec in zip(self.knn_clf.predict(faces_encodings), are_matches):
        if rec:
            #print (pred)
            print(confidence)
            return pred
        else:
            #print ("unknown")
            return "unknown"
```

Figure 19. Code for K-NN predictor

## 6 Experiments and Results

All experiments are tested using an Aftershock M15 notebook with NVIDIA GeForce GTX 950M 2GB display card and 16 GB RAM.

### 6.1 Experiment 1 – Face Detection

The objective of Experiment 1 is to prove that facial detection can affect the results of recognition algorithms. For this experiment, we will be using Dataset 1(Haar Cascade generated) with 47 identities, 50 training images and 10 test images per identity. There is a total of 2350 training images used and 470 images for testing. Identities that have lesser than 60 images in the dataset will not be used in this experiment. Haar Cascade detection and deep learning detection will be experimented on with LBPH recognition algorithm. Only 1 recognition algorithm is used to ensure fairness of the experiment. Figure 21 illustrates the results of Experiment 1.

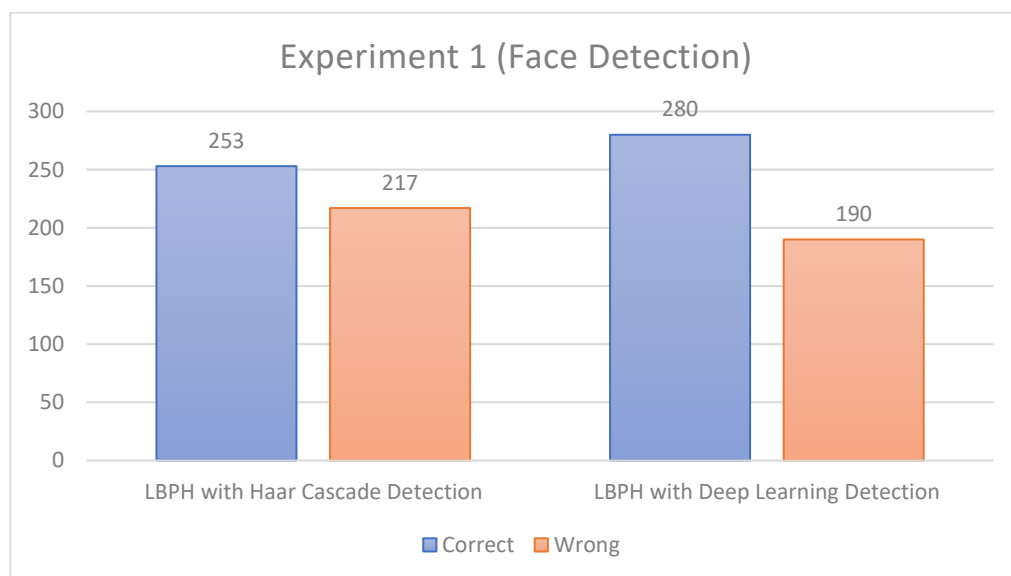


Figure 20.Results of Experiment 1

The dataset consists of images of complex settings and results for LBPH are of decent accuracy. However, this experiment focuses on face detection. The results show that LBPH with deep learning detection have better recognition results as compared to

LBPH with Haar Cascade detection, with an accuracy of 59.6% to 53.8% respectively. This indicates that face detection can affect the accuracy of recognition algorithms.

## 6.2 Experiment 2 – Generic Dataset

The objective of Experiment 2 is to compare the performances of LBPH algorithm and K-NN algorithm. This is to compare the general performances of the algorithms running on the dataset. Deep learning detection is used for both recognition algorithms.

Dataset 2 is selected for this experiment with 49 identities with 50 images used for training and 10 images for testing per identity. In total, 2450 training images used and 490 images for testing. Figure 22 illustrates the results of Experiment 2.

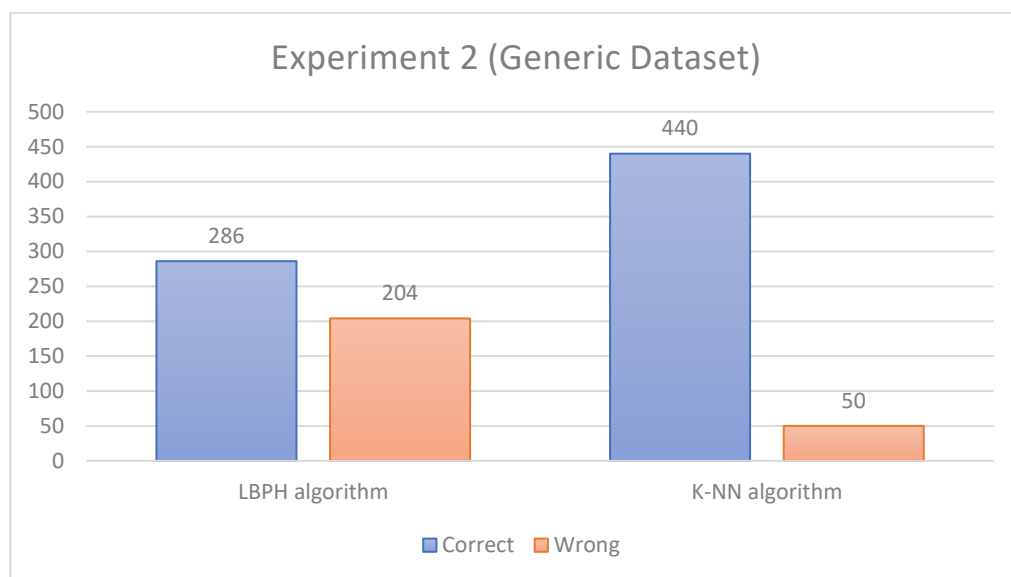


Figure 21. Results for Experiment 2

K-NN algorithm has better accuracy in recognition with an accuracy of 89.7% as compared to 58.4% accuracy for LBPH algorithm. LBPH is not giving a good recognition accuracy for this dataset and this dataset does not give a meaningful feedback regarding the factors that caused the lower accuracy.

### 6.3 Experiment 3 – Dataset with Clean Faces

With the results from experiment 3, we want to find out the factors that may affect both recognition algorithms. The objective of experiment 3 is to compare the performance of the 2 recognition algorithms with a “purer” dataset. It is to see if there is an improvement in recognition algorithms with test images of frontal faces, consistent lighting throughout the face and less noise on the face.

For this experiment, Dataset 3 is used, which consist of 20 identities, 40 training images and 10 testing images per identity. A total of 800 training images and 200 testing images are used. All identities have at least 50 valid images for training and testing. Figure 23 illustrate the results of Experiment 3.

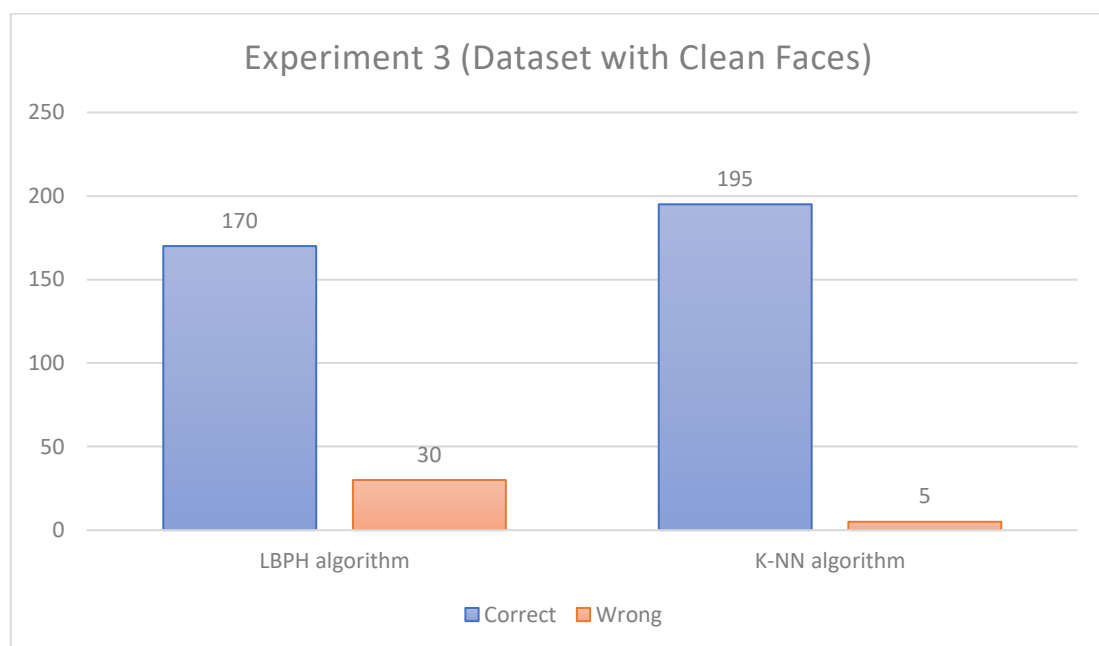


Figure 22. Results for Experiment 3

Both algorithms have improvement when recognizing using Dataset 3. LBPH have a recognition accuracy of 85% and K-NN have an accuracy of 97.5% . From this, we can see that LBPH performs relatively well in detecting frontal faces with less noise and consistent lighting on faces.

#### 6.4 Experiment 4 – Dataset with Darker Skin Tone

From the previous experiments, we know that lighting and face angle can affect recognition results of LBPH and K-NN. The purpose of Experiment 4 is to test the performance of algorithm in recognizing people with darker skin tones. Dataset 4 is used for this experiment with 21 identities, 50 images for training and 10 images for testing per identity. All identities have at least 60 valid images for training and testing. Since only frontal faces and consistent lighting of faces are used, we can rule out other factors and focus on skin tone. Figure 24 illustrate the results of Experiment 4.

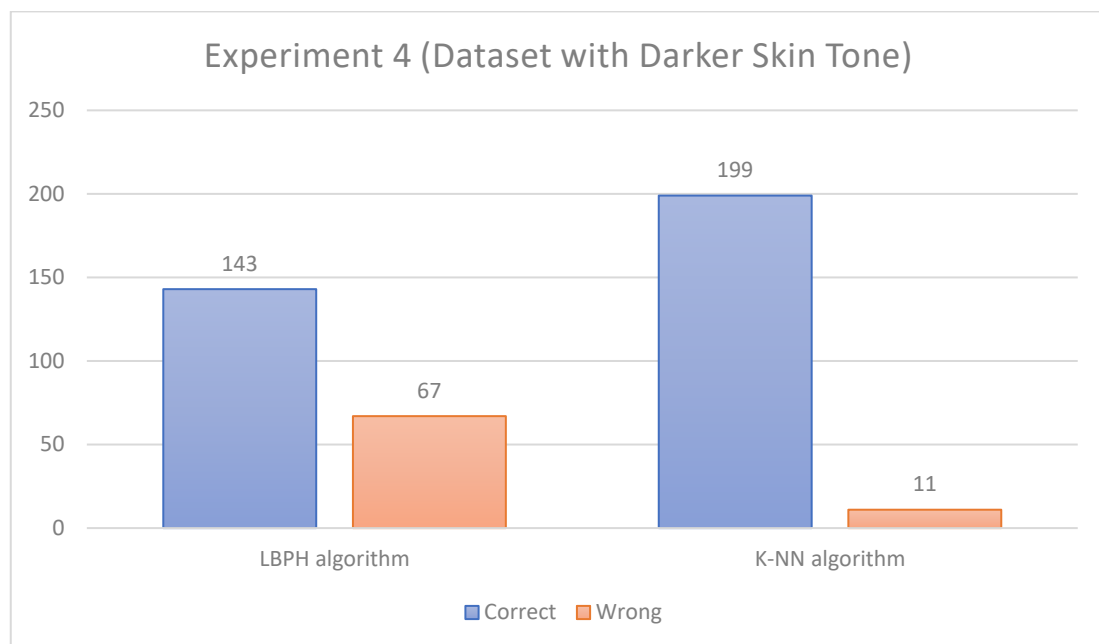


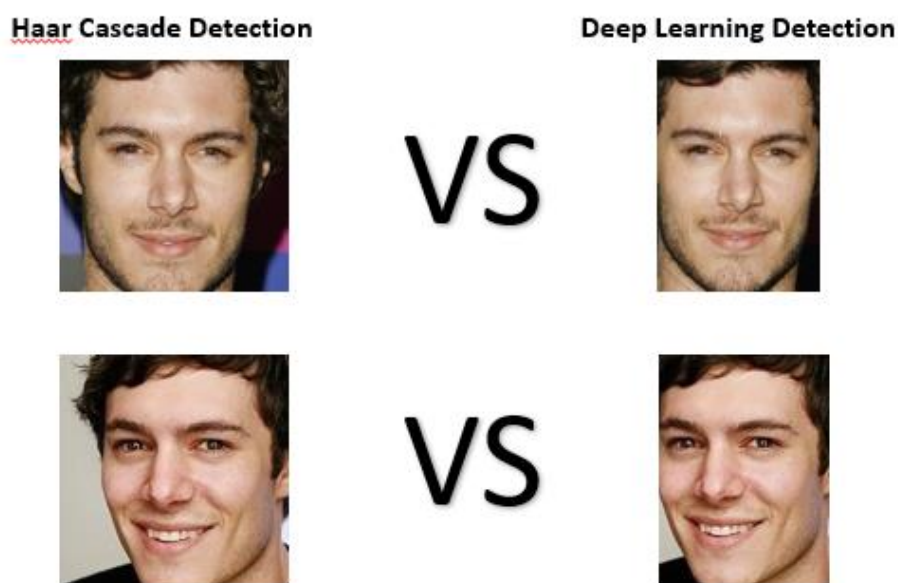
Figure 23. Results for Experiment 4

LBPH have an accuracy result of 68% and K-NN algorithm have an accuracy result of 94.8%. The results show that skin tone can affect recognition especially for LBPH as it suffers a significant drop in recognition accuracy.

## 7 Experiment Findings

### 7.1 Face Detection

Face detection can affect accuracy of recognition algorithm due to the difference in detecting and cropping faces. Haar Cascade algorithm has a wider cropping of face as compared to deep learning detection (refer to figure 25). Having a wider cropping area means adding more noise to the face. The additional space cropped can affect some recognition algorithms.



*Figure 24. Haar Cascade Detection vs Deep Learning Detection*

LBPH uses pixel intensity value of grayscale images to compute a histogram. Cropping a wider face means having more unwanted pixels being included in the calculated, generating a histogram which include unwanted features, affecting the results. One can argue that crop boxes can be resized to resolve this issue, but Haar Cascade algorithm's cropped faces may not consistently be at the centre. Some images cropped have more noise on one of the sides.



## 7.2 LBPH Algorithm

Overall, LBPH algorithm returns a fairly decent result when recognising faces. LBPH uses pixel intensity of grayscale image to process into a histogram for comparison. Having inconsistent lighting may affect the results of recognition as pixel value of features detected will be inconsistent between images, making features abstracted to be inaccurate. From experiment 3, we can see that 85% accuracy is achieved when all images used have consistent lighting and with frontal faces.

LBPH did not work as accurately when tested in experiment 4 as it is harder to distinguish the features of a face due to darker skin tone. This may be because LBPH converts images to grayscale to extract the features of faces and having a darker skin tone means there are lesser values to work with (generally lower values).

## 7.3 K-NN Algorithm

As compared to LBPH, KNN algorithm is significantly more accurate. However, the run time for KNN algorithm is much slower. This is because calling face\_recognition to return 128-d value takes time.

However, the recognition accuracy is much higher. Although the results in Experiment 2 shows 89% accuracy (50 wrong answers), most wrong answers are due to the face\_recognition not being able to get the value of 128d of the face, returning a NULL to the benchmarker. The images that cause NULL to be returned are being saved and analysed. The images have a few common factors. The factors are:

1. Faces are facing too much to one side (only half a face is shown),
2. Part of the face is covered by a hand or shadow, (only part of face is shown)
3. Angle of lighting (bright at the left and right of the image, but darker at the centre)

## 8 Conclusions

The datasets created for this project aims for a different approach when benchmarking face recognition algorithm. Most benchmarks provide datasets with images of complex factors to measure the performance of algorithms. This project wishes to break the complexity of these datasets, creating a more meaningful dataset. This allows programmers to understand more about their algorithms and find the strengths and weaknesses in dealing with different factors. Application specific facial recognition algorithm can also be realised using these datasets. These meaningful datasets aim to become an essential benchmark for all facial recognition algorithms.

## 9 Recommendations

The datasets generated have a limited number of images and identities. More identities and images can be added to the dataset to have a better and fairer analysis of the algorithm. More meaningful datasets can be created by breaking the datasets into more factors (applied makeup, etc.). This project hopes to become an open source project (GitHub) where contributors can contribute to this project. The dataset needs revision as images are filtered by the author and therefore, bias to the author's opinion.

## References

- [1] J. Lee, “Singapore government researching facial recognition applications,” 06 11 2017. [Online]. Available: <http://www.biometricupdate.com/201711/singapore-government-researching-facial-recognition-applications>. [Accessed 04 2018].
- [2] H. Makwana and T. Singh, “Comparison of Different Algorithm for Facial Recognition,” *Global Journals Inc. USA*, vol. 13, 2013.
- [3] A. J. O’Toole, P. Philips, F. Jiang, J. Ayyad, N. Penard and H. Abdi, “Face Recognition Algorithms Surpass Humans,” National Institute of Standards and Technology, 2016.
- [4] L. Goode, “Facial recognition software is biased towards white men, researcher finds,” *The Verge*, 11 02 2018. [Online]. Available: <https://www.theverge.com/2018/2/11/17001218/facial-recognition-software-accuracy-technology-mit-white-men-black-women-error>. [Accessed 20 09 2018].
- [5] J. Hui, “SSD object detection: Single Shot MultiBox Detector for real-time processing,” 14 March 2018. [Online]. Available: [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06). [Accessed 25 September 2018].
- [6] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller and E. Brossard, “The MegaFace Benchmark: 1 Million Faces for recognition at Scale,” Department of Computer Science and Engineering, University of Washington, 2016.
- [7] A. Nech, I. Kemelmacher-Shlizerman and P. G. Allen, “Level Playing Field for Million Scale Face Recognition,” School of Computer Science and Engineering, University of Washington, 2017.
- [8] G. B. Huang and E. Learned-Miller, “Labeled Faces in the Wild: Updates and New Reporting Procedures,” University of Massachusetts, Boston, 2014.
- [9] H.-W. Ng and S. Winkler, “A DATA-DRIVEN APPROACH TO CLEANING LARGE FACE DATASETS,” Advanced Digital Sciences Center (ADSC), University of Illinois at Urbana-Champaign, Singapore, 2014.

## Program Listing

*benchmarker.py*

```
import os
import random
import cv2
from PIL import Image
import numpy as np
import shutil
import importlib
import imp

testset_amount = 10
selected_Dataset = None
trainingSeq = 0
testSeq = 0
test_Question = []
test_Dir = []
bm = None
MajorDataset = "Dataset 5"

def fetchTrainingData(DS_DIR, TRAIN_DIR = "Training Image"):
    if DS_DIR == MajorDataset and TRAIN_DIR == "Training Image":
        global trainingSeq
        if trainingSeq == 0:
            trainingSeq = 1
            return fetchTrain(DS_DIR, "Training-Pure")

        elif trainingSeq == 1:
            trainingSeq = 2
            return fetchTrain(DS_DIR, "Training-Mix")

    else:
        return fetchTrain(DS_DIR, TRAIN_DIR)

def fetchTestQuestion(test_Name = "Test Image"):
    if selected_Dataset == MajorDataset and test_Name == "Test Image":
        global testSeq
        if testSeq == 0:
            testArr = fetchTest("Test 1")
            testSeq = 1
            return testArr

        elif testSeq == 1:
            testSeq = 2
            return fetchTest("Test 2 - Angle")

        elif testSeq == 2:
            testSeq = 3
            return fetchTest("Test 3 - Lighting")

    else:
        return fetchTest(test_Name)

def fetchTest(test_Name = "Test Image"):
    global test_Question
    global test_Dir

    if selected_Dataset is not None:
        DS_DIR = selected_Dataset

        test_Dir.clear()
        test_Question.clear()
        test_set = []
```

```

    print("*** Fetching Test Images... **")
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    img_dir = os.path.join(BASE_DIR, DS_DIR)
    testset_data = os.path.join(img_dir, test_Name)

    total_files = len(os.listdir(testset_data)) * testset_amount
    rand = random.sample(range(0, total_files), total_files)

    for ran in rand:
        iden_id = int(ran / testset_amount)
        img_no = ran % testset_amount

        iden_dir = os.path.join(testset_data,
os.listdir(testset_data)[iden_id])
        img = os.listdir(iden_dir)[img_no]
        testing_dir = os.path.join(iden_dir, img)
        image = cv2.imread(testing_dir)
        test_Dir.append(testing_dir)
        test_set.append(image)
        test_Question.append(os.listdir(testset_data)[iden_id].replace("
", "-").lower())
        #name = bm.testAlgo(image, DS_DIR)

    print("*** Fetch Completed **")
    return test_set

def submitAnswer(ansArr):
    global test_Question
    global test_Dir
    correctAns = 0
    wrongAns = 0

    if not test_Question:
        print("Please fetch the test questions before submitting")
    else:
        BASE_DIR = os.path.dirname(os.path.abspath(__file__))
        wrong_dir = os.path.join(BASE_DIR, "Incorrect Answer")

        if not os.path.exists(wrong_dir):
            os.makedirs(wrong_dir)
        else:
            shutil.rmtree(wrong_dir)
            os.makedirs(wrong_dir)

        print("*** Checking your answer **")
        for x in range(len(test_Question)):
            print("Question: " + test_Question[x].replace(" ", "-").lower())
            if ansArr[x] is not None:
                print("Answer: " + ansArr[x].replace(" ", "-").lower())
            else:
                print("Answer: ")
                print("")
            if ansArr[x] is not None and ansArr[x].replace(" ", "-").lower()
== test_Question[x]:
                correctAns += 1
            else:
                wrongAns += 1
                copyDir = str(x) + " Qn-" + test_Question[x] + " Ans-" +
ansArr[x].replace(" ", "-").lower()
                shutil.copyfile(test_Dir[x], (wrong_dir + "\\\" +
str(copyDir)))

        print("No of correct: " + str(correctAns))

```

```

print("No of wrong: " + str(wrongAns))
acc = (correctAns / (correctAns + wrongAns)) * 100
print("Accuracy is " + str(acc) + "%\n")
return correctAns, wrongAns, acc

def feedTestData(DS_DIR, TEST_DIR="Test Image"):
    print("**Initiating Test, calling testAlgo() method **")
    try:
        correctAns = 0
        wrongAns = 0
        BASE_DIR = os.path.dirname(os.path.abspath(__file__))
        img_dir = os.path.join(BASE_DIR, DS_DIR)
        testset_data = os.path.join(img_dir, TEST_DIR)
        wrong_dir = os.path.join(BASE_DIR, "Incorrect Answer")

        if not os.path.exists(wrong_dir):
            os.makedirs(wrong_dir)
        else:
            shutil.rmtree(wrong_dir)
            os.makedirs(wrong_dir)

        print("Total Number of test = " + str(len(os.listdir(testset_data))))
        total_files = len(os.listdir(testset_data)) * testset_amount
        rand = random.sample(range(0, total_files), total_files)

        for ran in rand:
            iden_id = int(ran / testset_amount)
            img_no = ran % testset_amount

            iden_dir = os.path.join(testset_data,
os.listdir(testset_data)[iden_id])
            img = os.listdir(iden_dir)[img_no]
            testimg_dir = os.path.join(iden_dir, img)
            image = cv2.imread(testimg_dir)
            name = bm.testAlgo(image, DS_DIR)
            print("Question: " + os.listdir(testset_data)[iden_id].replace("
", "-").lower())
            if name is not None:
                print("Answer: " + name.replace(" ", "-").lower())
            else:
                print("Answer: ")
                print("")
            if name is not None and name.replace(" ", "-").lower() ==
os.listdir(testset_data)[iden_id].replace(" ", "-").lower():
                correctAns += 1
            else:
                wrongAns += 1
                shutil.copyfile(testimg_dir, (wrong_dir + "\\" + img))

        print("No of correct: " + str(correctAns))
        print("No of wrong: " + str(wrongAns))
        acc = (correctAns / (correctAns+wrongAns))*100
        print("Accuracy is " + str(acc) + "%\n")

        return correctAns, wrongAns, acc

    except Exception as e:
        print("Please ensure you code have a method name testAlgo(image)")
        print(e)

def fetchTrain(DS_DIR, TRAIN_DIR = "Training Image"):
    try:
        imageArr = []

```

```

labelArr = []
global selected_Dataset
selected_Dataset = DS_DIR

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
img_dir = os.path.join(BASE_DIR, DS_DIR)
training_data = os.path.join(img_dir, TRAIN_DIR)

print("Preparing images for training...")

for root, dirs, files in os.walk(training_data):
    for file in files:
        if file.lower().endswith("png") or
file.lower().endswith("jpg") or file.lower().endswith("jpeg"):
            path = os.path.join(root, file)
            label =
os.path.basename(os.path.dirname(path)).replace(" ", "-").lower()
            image = cv2.imread(path)
            imageArr.append(image)
            labelArr.append(label)

        #bm.testAlgo(imageArr)
        return imageArr, labelArr, DS_DIR
        #bm.trainAlgo(imageArr,labelArr ,DS_DIR)
except Exception as e:
    print(e)
    #print("Please ensure you code have a method name
trainAlgo(imageArray[], label[], DS_NAME)")

    return None

def main():
    pythonFile = input("Key in your ALGORITHM file name (without .py): ")
    try:
        global bm
        bm = importlib.import_module(pythonFile, ".")
        menu = True
        spam_info = imp.find_module(pythonFile)
        #print(spam_info)
        print("Import ALGORITHM FILE successful")

        while True:
            print("Dataset to generate Train and Test Set:")
            print("1) Dataset 1")
            print("2) Dataset 2")
            print("3) Dataset 3")
            print("4) Dataset 4")
            print("5) Dataset 5")
            print("0) Exit")

            try:
                choice = int(input("Your Choice: "))
                if choice == 1:
                    print("**Initiating training, calling trainAlgo()
method.**")

                    print("")
                    imageArr, labelArr, DS_DIR = fetchTrainingData("Dataset
1")

                    bm.trainAlgo(imageArr, labelArr, DS_DIR)
                    feedTestData("Dataset 1")
                elif choice == 2:
                    print("**Initiating training, calling trainAlgo()
method.**")

```

```

        print("")
        imageArr, labelArr, DS_DIR = fetchTrainingData("Dataset
2")

        bm.trainAlgo(imageArr, labelArr, DS_DIR)
        feedTestData("Dataset 2")
    elif choice == 3:
        print("**Initiating training, calling trainAlgo()
method.**")

        print("")
        imageArr, labelArr, DS_DIR = fetchTrainingData("Dataset
3")

        bm.trainAlgo(imageArr, labelArr, DS_DIR)
        feedTestData("Dataset 3")
    elif choice == 4:
        print("**Initiating training, calling trainAlgo()
method.**")

        print("")
        imageArr, labelArr, DS_DIR = fetchTrainingData("Dataset
4")

        bm.trainAlgo(imageArr, labelArr, DS_DIR)
        feedTestData("Dataset 4")
    elif choice == 5:
        print("**Initiating training(pure face images), calling
trainAlgo() method.**")
        print("")

        imageArr, labelArr, DS_DIR =
fetchTrainingData(MajorDataset, "Training-Pure")
        bm.trainAlgo(imageArr, labelArr, DS_DIR)
        print("Test 1 Started:")
        correctAns, wrongAns, acc = feedTestData("Dataset
Mix", "Test 1")

        print("\n Initiating training 2(mix factor images),
calling trainAlgo() method")
        imageArr2, labelArr2, DS_DIR2 =
fetchTrainingData(MajorDataset, "Training-Mix")
        bm.trainAlgo(imageArr2, labelArr2, DS_DIR2 + "2")

        print("Test 2 Started: ")
        correctAns2, wrongAns2, acc2 =
feedTestData(MajorDataset, "Test 2 - Angle")
        print("Test 3 Started: ")
        correctAns3, wrongAns3, acc3 =
feedTestData(MajorDataset, "Test 3 - Lighting")

        print("===== Test 1 (Pure Faces) Results
=====")

        print ("No of correct answer: " + str(correctAns))
        print ("No of wrong answer: " + str(wrongAns))
        print ("Accuracy: " + str(acc))

        print("")
        print("===== Test 2 (Faces of different angle)
Results =====")

        print ("No of correct answer: " + str(correctAns2))
        print ("No of wrong answer: " + str(wrongAns2))
        print ("Accuracy: " + str(acc2))

        print("")
        print("===== Test 3 (Faces of different
lighting) Results =====")

        print ("No of correct answer: " + str(correctAns3))
        print ("No of wrong answer: " + str(wrongAns3))
        print ("Accuracy: " + str(acc3))

```



```

        elif choice == 0:
            print("Exiting ...")
            break
        else:
            print("Invalid input.")
    except Exception as e:
        print(e)
        print("Please check you have selected the correct Dataset")

except Exception as e:
    print(e)
    print("Fail to import ALGORITHM file. Please check that ")

if __name__ == "__main__":
    main()

```

*lbph\_algo.py*

```

import os
import cv2
import sys
import PIL
from PIL import Image #python image library
import numpy as np
import pickle
import lbph_predict as lp
import importlib
import imp

at = lp.algorithm_test()

def testAlgo(image, DS_DIR):
    if at.get_yamlfile() is None:
        at.set_yamlfile(DS_DIR)

    if at.get_pickFile() is None:
        at.set_pickFile(DS_DIR)

    return at.lbph_pred(image)

def trainAlgo(imageArr, labelArr, DIR_NAME):

    proto = "ML/deploy.prototxt.txt"
    caffmodel = "ML/res10_300x300_ssd_iter_140000.caffemodel"
    confid = 0.7

    net = cv2.dnn.readNetFromCaffe(proto, caffmodel)
    recognizer = cv2.face.LBPHFaceRecognizer_create()

    current_id = 0
    label_ids = {} #dictionary
    y_labels = []
    x_train = []

    for x in range(len(imageArr)):
        if not labelArr[x] in label_ids:
            label_ids[labelArr[x]] = current_id
            current_id += 1

```

```

        id_ = label_ids[labelArr[x]]
        #print(label_ids)

    #pil_image = Image.open(path).convert("L") #grayscale
    try:

        image = np.array(imageArr[x])
        (h, w) = image.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)),
1.0, (300, 300), (104.0, 177.0, 123.0))

        net.setInput(blob)
        detections = net.forward()

        for i in range(0, detections.shape[2]):
            #print(detections.shape)

            confidence = detections[0, 0, i, 2]
            if confidence > confid:

                box = detections[0, 0, i, 3:7] * np.array([w, h, w,
h])

                #print(box)
                (startX, startY, endX, endY) = box.astype("int")
                roi = image[startY:endY, startX:endX]
                #roi = roi.cvt
                try:
                    if cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY).shape:
                        x_train.append(cv2.cvtColor(roi,
cv2.COLOR_BGR2GRAY)) #why do this is to save multiple person.
                        y_labels.append(id_)
                except Exception as e:
                    pass
                #print((str(labelArr[x]) + " " + str(current_id)), end = "",
flush = True)
                sys.stdout.write("\r" + str(x+1) + " of " +
str(len(imageArr)) + " has been processed");
                sys.stdout.flush()
            except Exception as e:
                print(e)

        x_train = normalize_img(x_train)
        x_train = resize(x_train)
        with open(str(DIR_NAME) + "_LBPH.pickle", 'wb') as f: #wb = writing byte
            pickle.dump(label_ids, f)
        recognizer.train(x_train, np.array(y_labels))
        recognizer.write(str(DIR_NAME) + "_LBPH.yml")
        print("\n **LBPH Training Completed**\n")

def normalize_img(images):
    images_normalized = []
    for image in images:
        try:
            if len(image.shape) == 3:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                images_normalized.append(cv2.equalizeHist(image))
            except Exception as e:
                pass
        return images_normalized

def resize(images, size = (100,100)):
    images_norm = []
    for image in images:
        if image.shape < size:
            image_norm = cv2.resize(image, size, interpolation =

```

```

cv2.INTER_AREA)
    else:
        image_norm = cv2.resize(image, size,
interpolation=cv2.INTER_CUBIC)

        images_norm.append(image_norm)
    return images_norm

def main():
    pythonFile = "benchmarker"
    nameList = []
    nameList.clear()
    try:
        bm = importlib.import_module(pythonFile, ".")
        menu = True
        spam_info = imp.find_module(pythonFile)
        print("Import Benchmark successful")

        ##### Dataset 1-4#####

        DS_DIR = "Dataset 4"
        imageArr, labelArr, DS_DIR = bm.fetchTrainingData(DS_DIR)
        print("Run Successful")
        trainAlgo(imageArr, labelArr, DS_DIR)

        imgArr = bm.fetchTestQuestion()
        #print(len(imgArr))
        for i in range(len(imgArr)):
            name = testAlgo(imgArr[i], DS_DIR)
            nameList.append(name)

        bm.submitAnswer(nameList)

        ##### Dataset 5 #####

        DS_DIR = "Dataset 5"
        imageArr, labelArr, DS = bm.fetchTrainingData(DS_DIR)
        print("Run Successful")
        trainAlgo(imageArr, labelArr, DS)

        imgArr = bm.fetchTestQuestion()
        #print(len(imgArr))
        for i in range(len(imgArr)):
            name = testAlgo(imgArr[i], DS)
            nameList.append(name)

        correctAns1, wrongAns1, acc1 = bm.submitAnswer(nameList)
        nameList.clear()
        ### Training for Mixed Training
        imageArr2, labelArr2, DS_DIR2 = bm.fetchTrainingData(DS_DIR)
        print("Run Successful 2")

        trainAlgo(imageArr2, labelArr2, DS_DIR2+"2")# +"2" is to save the
file under different name. If u want to replace the previous file, you need
not +"2"
        ### Fetching 2nd test - Angle
        imgArr2 = bm.fetchTestQuestion()

        for i in range(len(imgArr)):
            name = testAlgo(imgArr2[i], DS_DIR2+"2")
            nameList.append(name)

        correctAns2, wrongAns2, acc2 = bm.submitAnswer(nameList)
        nameList.clear()

```

```

    ### Fetching 3rd Test - Lighting
    imgArr3 = bm.fetchTestQuestion()
    # print(len(imgArr))
    for i in range(len(imgArr3)):
        name = testAlgo(imgArr3[i], DS_DIR2+"2")
        nameList.append(name)

    correctAns3, wrongAns3, acc3 = bm.submitAnswer(nameList)
    nameList.clear()
    print("===== Test 1 (Pure Faces) Results
=====")
    print ("No of correct answer: " + str(correctAns1))
    print ("No of wrong answer: " + str(wrongAns1))
    print ("Accuracy: " + str(acc1))

    print("")
    print("===== Test 2 (Faces of different angle) Results
=====")
    print ("No of correct answer: " + str(correctAns2))
    print ("No of wrong answer: " + str(wrongAns2))
    print ("Accuracy: " + str(acc2))

    print("")
    print("===== Test 3 (Faces of different lighting) Results
=====")
    print ("No of correct answer: " + str(correctAns3))
    print ("No of wrong answer: " + str(wrongAns3))
    print ("Accuracy: " + str(acc3))

except Exception as e:
    print (e)

if __name__ == "__main__":
    main()

```

### lbph\_predict.py

```

import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import pickle

class algorithm_test:

    def __init__(self):
        caffmodel = "ML/res10_300x300_ssd_iter_140000.caffemodel"
        proto = "ML/deploy.prototxt.txt"
        self.confid = 0.7
        self.net = cv2.dnn.readNetFromCaffe(proto, caffmodel)
        self.ymlfile = None
        self.recognizer = cv2.face.LBPHFaceRecognizer_create()
        self.labels = {}
        self.pickFile = None

    def set_ymlfile(self, ymlfile):
        self.ymlfile = str(ymlfile)+"_LBPH.yml"
        self.recognizer.read(self.ymlfile)

    def set_pickFile(self, pickFile):

```

```

self.pickFile = str(pickFile)+"_LBPH.pickle"
with open(self.pickFile, 'rb') as f:
    self.labels = pickle.load(f)
    #print(self.labels)
    self.labels = {v:k for k,v in self.labels.items()}

def get_pickFile(self):
    return self.pickFile

def get_ymlfile(self):
    return self.ymlfile

def lbph_pred(self, img):
    name = ""
    roi_gray = []

    image = np.array(img)
    (h, w) = image.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
(300, 300), (104.0, 177.0, 123.0))

    self.net.setInput(blob)
    detections = self.net.forward()

    for i in range(0, detections.shape[2]):
        # print(detections.shape)
        confidence = detections[0, 0, i, 2]

        if confidence > self.confid:
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            try:
                (cv2.cvtColor(image[startY:endY, startX: endX],
cv2.COLOR_BGR2GRAY))
                roi_gray.append(cv2.cvtColor(image[startY:endY, startX:
endX], cv2.COLOR_BGR2GRAY))

                roi_gray = self.normalize_img(roi_gray)
                roi_gray = self.resize(roi_gray)

                id_, conf = self.recognizer.predict(roi_gray[0])
                print("confidence: " + str(conf))
                if conf >= 0 and conf <= 120:
                    name = self.labels[id_]
                    #print(name)
                    #print(self.labels[id_])
            except Exception as e:
                pass

    return name

def normalize_img(self, images):
    images_normalized = []
    for image in images:
        if len(image.shape) == 3:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            images_normalized.append(cv2.equalizeHist(image))
    return images_normalized

def resize(self, images, size=(100, 100)):
    images_norm = []
    for image in images:
        if image.shape < size:
            image_norm = cv2.resize(image, size,

```

```

interpolation=cv2.INTER_AREA)
        else:
            image_norm = cv2.resize(image, size,
interpolation=cv2.INTER_CUBIC)

            images_norm.append(image_norm)
    return images_norm

```

### *knn\_algo.py*

```

import importlib
import imp
import math
from sklearn import neighbors
import os
import sys
import pickle
from PIL import Image, ImageDraw
import numpy as np
import face_recognition
from face_recognition.face_recognition_cli import image_files_in_folder
import cv2
import knn_predict as fra

knp = fra.knn_prediction()
# ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
# BASE_DIR = os.path.dirname(os.path.abspath(__file__))
# img_dir = os.path.join(BASE_DIR, "../images")
# training_data = os.path.join(img_dir, "Training Image")

# def testAlgo(img):
#     #pil_image = Image.open('Image.jpg').convert('RGB')
#     #print(img[3])
#     open_cv_image = np.array(img[5])
#     imageRGB = cv2.cvtColor(open_cv_image, cv2.COLOR_BGR2RGB)
#     open_cv_image.shape[:2]
#     print(len(img))
#     cv2.imshow('image', open_cv_image)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()
def testAlgo(image , DIR_NAME):

    if knp.get_model_path() is None:
        knp.set_model_path(DIR_NAME)

    return knp.predict(image)

def trainAlgo(imageArr,labelArr, DIR_NAME):

    X_train = []
    y_labels = []
    model_save_path = str(DIR_NAME) + "_knn.clf"
    n_neighbors = 3
    #model_save_path = None
    #n_neighbors = None
    knn_algo = 'ball_tree'
    verbose = False

    proto = "ML/deploy.prototxt.txt"
    caffmodel = "ML/res10_300x300_ssd_iter_140000.caffemodel"

```

```

confid = 0.7

net = cv2.dnn.readNetFromCaffe(proto, caffmodel)

for x in range(len(imageArr)):
    #print("Training Identity " + labelArr[x] + " " + str(x))
    sys.stdout.write("\r" + str(x + 1) + " of " + str(len(imageArr))
+ " has been processed");
    sys.stdout.flush()
    try:
        count = 0
        imageA = np.array(imageArr[x])
        #imageRGB = cv2.cvtColor(imageA, cv2.COLOR_BGR2RGB)
        (h, w) = imageA.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(imageA, (300, 300)),
1.0, (300, 300), (104.0, 177.0, 123.0))

        net.setInput(blob)
        detections = net.forward()

        for i in range(0, detections.shape[2]):

            # print(detections.shape)
            count += 1
            confidence = detections[0, 0, i, 2]
            if confidence > confid and count == 1:
                box = detections[0, 0, i, 3:7] * np.array([w, h,
w, h])

                (startX, startY, endX, endY) = box.astype("int")
                #face_bounding_boxes =
                ("+"startX+", "+"endX+", "+"startY+", "+"endY+")"
                roi = imageA[startY:endY, startX: endX]
                #print(face_recognition.face_encodings(roi))

X_train.append(face_recognition.face_encodings(roi)[0])
y_labels.append(labelArr[x])

    except Exception as e:
        print ("")
        print (e)

if n_neighbors is None:
    n_neighbors = int(round(math.sqrt(len(X))))
    if verbose:
        print("Chose n_neighbors automatically:", n_neighbors)

    # Create and train the KNN classifier
    knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors,
algorithm=knn_algo, weights='distance')
    knn_clf.fit(X_train, y_labels)

    # Save the trained KNN classifier
    if model_save_path is not None:
        with open(model_save_path, 'wb') as f:
            pickle.dump(knn_clf, f)
            print("***Training Completed**")

    return knn_clf

def main():
    pythonFile = "benchmarker"
    nameList = []
    nameList.clear()
    try:
        bm = importlib.import_module(pythonFile, ".")

```

```

menu = True
spam_info = imp.find_module(pythonFile)
print("Import Benchmark successful")

##### Dataset 1-4#####

# DS_DIR = "Dataset 4"
# imageArr, labelArr, DS_DIR = bm.fetchTrainingData(DS_DIR)
# print("Run Successful")
# trainAlgo(imageArr, labelArr, DS_DIR)
#
# imgArr = bm.fetchTestQuestion()
# #print(len(imgArr))
# for i in range(len(imgArr)):
#     name = testAlgo(imgArr[i], DS_DIR)
#     nameList.append(name)
#
# bm.submitAnswer(nameList)

##### Dataset 5 #####

DS_DIR = "Dataset 5"
imageArr, labelArr, DS = bm.fetchTrainingData(DS_DIR)
print("Run Successful")
trainAlgo(imageArr, labelArr, DS)

imgArr = bm.fetchTestQuestion()
#print(len(imgArr))
for i in range(len(imgArr)):
    name = testAlgo(imgArr[i], DS)
    nameList.append(name)

correctAns1, wrongAns1, acc1 = bm.submitAnswer(nameList)
nameList.clear()
### Training for Mixed Training
imageArr2, labelArr2, DS_DIR2 = bm.fetchTrainingData(DS_DIR)
print("Run Successful 2")

trainAlgo(imageArr2, labelArr2, DS_DIR2+"2")# +"2" is to save the
file under different name. If u want to replace the previous file, you need
not +"2"
### Fetching 2nd test - Angle
imgArr2 = bm.fetchTestQuestion()

for i in range(len(imgArr)):
    name = testAlgo(imgArr2[i], DS_DIR2+"2")
    nameList.append(name)

correctAns2, wrongAns2, acc2 = bm.submitAnswer(nameList)
nameList.clear()
### Fetching 3rd Test - Lighting
imgArr3 = bm.fetchTestQuestion()
# print(len(imgArr))
for i in range(len(imgArr3)):
    name = testAlgo(imgArr3[i], DS_DIR2+"2")
    nameList.append(name)

correctAns3, wrongAns3, acc3 = bm.submitAnswer(nameList)
nameList.clear()
print("===== Test 1 (Pure Faces) Results
=====")
print ("No of correct answer: " + str(correctAns1))
print ("No of wrong answer: " + str(wrongAns1))
print ("Accuracy: " + str(acc1))

```



```

        print("")
        print("===== Test 2 (Faces of different angle) Results
=====")
        print ("No of correct answer: " + str(correctAns2))
        print ("No of wrong answer: " + str(wrongAns2))
        print ("Accuracy: " + str(acc2))

        print("")
        print("===== Test 3 (Faces of different lighting) Results
=====")
        print ("No of correct answer: " + str(correctAns3))
        print ("No of wrong answer: " + str(wrongAns3))
        print ("Accuracy: " + str(acc3))

    except Exception as e:
        print (e)

if __name__ == "__main__":
    main()

```

### *knn\_predict.py*

```

import math
from sklearn import neighbors
import os
import os.path
import pickle
from PIL import Image, ImageDraw
import numpy as np
import face_recognition
from face_recognition.face_recognition_cli import image_files_in_folder
import cv2
import shutil

filename = 0
class knn_prediction:

    def __init__(self):
        proto = "ML/deploy.prototxt.txt"
        caffmodel = "ML/res10_300x300_ssd_iter_140000.caffemodel"
        self.model_path = None
        self.knn_clf = ""

        self.confid = 0.7
        self.net = cv2.dnn.readNetFromCaffe(proto, caffmodel)
        self.distance_threshold = 0.6

    def set_model_path(self, model_path):
        self.model_path = str(model_path)+"_knn.clf"
        with open(self.model_path, 'rb') as f:
            self.knn_clf = pickle.load(f)

    def get_model_path(self):
        return self.model_path

    def predict (self, img):

```

```

        image = np.array(img)
        (h, w) = image.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
(300, 300), (104.0, 177.0, 123.0))

        self.net.setInput(blob)
        detections = self.net.forward()

        for i in range(0, detections.shape[2]):
            # print(detections.shape)
            confidence = detections[0, 0, i, 2]

            if confidence > self.confid:
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                try:
                    faces_encodings =
face_recognition.face_encodings(image[startY:endY, startX: endX])
                    #print(len(faces_encodings))
                    # Use the KNN model to find the best matches for the
test face

                    closest_distances =
self.knn_clf.kneighbors(faces_encodings, n_neighbors=1)
                    are_matches = [closest_distances[0][0][0] <=
self.distance_threshold]
                    # Predict classes and remove classifications that aren't
within the threshold
                    for pred, rec in
zip(self.knn_clf.predict(faces_encodings), are_matches):
                        if rec:
                            #print (pred)
                            print(confidence)
                            return pred
                        else:
                            #print ("unknown")
                            return "unknown"
                    #return [(pred, loc) if rec else ("unknown", loc) for
pred, loc, rec in zip(knn_clf.predict(faces_encodings), X_face_locations,
are_matches)]

                except Exception as e:
                    print ("No 128d returned")

                    global filename
                    filename += 1

                    file = str(filename) + ".jpg"
                    # identityName =
os.path.basename(os.path.dirname(img)).replace(" ", "-").lower()
                    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
                    img_fold = os.path.join(BASE_DIR, "error images")

                    if not os.path.exists(img_fold):
                        os.makedirs(img_fold)

                    #
                    # identity_fold = os.path.join(img_fold, identityName)
                    # if not os.path.exists(identity_fold):
                    #     os.makedirs(identity_fold)
                    #
                    cv2.imwrite(os.path.join(img_fold, file),
image[startY:endY, startX: endX])

```

```
return "error"
```

### *TrainTest\_Generator.py*

```
import os
import shutil

def main():
    while True:
        print("Dataset to generate Train and Test Set:")
        print("1) Dataset 1")
        print("2) Dataset 2")
        print("3) Dataset 3")
        print("4) Dataset 4")
        print("0) Exit")

        try:
            choice = int(input("Your Choice: "))
            if choice == 1:
                generateTrain_Folder("Dataset 1")
            elif choice == 2:
                generateTrain_Folder("Dataset 2")
            elif choice == 3:
                generateTrain_Folder("Dataset 3", 40, 10)
            elif choice == 4:
                generateTrain_Folder("Dataset 4")
            elif choice == 0:
                break
            else:
                print("Invalid input.")
        except Exception as e:
            print(e)
            print("Please check you have selected the correct Dataset")
        print("Exiting ...")

def generateTrain_Folder(DS_DIR, trainSetNo = 50, testSetNo = 10):

    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    img_dir = os.path.join(BASE_DIR, DS_DIR)
    original_Data = os.path.join(img_dir, "dataset")
    training_data = os.path.join(img_dir, "Training Image")
    test_data = os.path.join(img_dir, "Test Image")

    trainingset_amount = trainSetNo
    test_set_amount = testSetNo
    counter = 1

    if not os.path.exists(training_data):
        os.makedirs(training_data)
    if not os.path.exists(test_data):
        os.makedirs(test_data)
    # print (os.listdir(original_Data))
    for iden in os.listdir(original_Data):
        print(iden)
        img_folders = os.path.join(original_Data, iden)
        counter = 1
        for root, dirs, files in os.walk(img_folders):
            img_counter = len(files)
            # print(files)
            # print(img_counter)
            identity = os.path.basename(os.path.dirname(os.path.join(root,
```

```

files[0])).replace(" ", "-").lower()
    if (img_counter < trainingset_amount + test_set_amount):
        print(
            "The identity " + identity + " have too little image to
train and test. The minimum image required is " + str(
                trainingset_amount + test_set_amount))
    else:
        for file in files:
            directory = os.path.join(root, file)
            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                identityName =
os.path.basename(os.path.dirname(directory)).replace(" ", "-").lower()
                if counter <= trainingset_amount:
                    training_folder = os.path.join(training_data,
identityName)

                    if not os.path.exists(training_folder):
                        os.makedirs(training_folder)
                    shutil.copy(directory, training_folder + "\\\" +
file)

                    counter += 1

                elif counter > trainingset_amount and counter <=
trainingset_amount + test_set_amount:
                    test_folder = os.path.join(test_data,
identityName)

                    if not os.path.exists(test_folder):
                        os.makedirs(test_folder)
                    shutil.copy(directory, test_folder + "\\\" +
file)

                    counter += 1

                elif counter >= trainingset_amount +
test_set_amount:
                    counter = 1
                    print(str(identityName) + " have been sorted
successfully")
                    break

if __name__ == "__main__":
    main()

```